Generalization of Single-Center Projections Using Projection Tile Screens

Matthias Trapp & Jürgen Döllner

Hasso-Plattner-Institute, University of Potsdam,
{matthias.trapp | juergen.doellner}@hpi.uni-potsdam.de
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany

Abstract. This work presents an image-based approach to efficiently generate multiple non-planar projections of arbitrary 3D scenes in real-time. The creation of projections such as panorama or fisheye views has a number of possible applications, e.g., in geovirtual environments and in augmented reality. Our rendering technique is based on dynamically created cube map textures in combination with shader programs that calculate the specific projections. Based on this principle, we present an approach to customize and combine different planar as well as non-planar projections. Our technique can be applied within a single rendering pass, is easy to implement, and exploits the capability of modern programmable graphics hardware completely.

1 Introduction

This work focuses on an image-based concept to compensate the field-of-view (FOV) limitations of the classical pinhole camera rendering pipeline. It has been developed to enable the application of non-planar projection on standard consumer graphics hardware in real-time. Examples are omni-directional panorama for non-planar screens or spherical dome projections [6]. This work focuses on real-time modifications of perspective views that are possible due to the recent hardware developments [4]. Our approach is limited to SCOP. In spite of non-planar projections screens [25], this technique can also be used for rendering effects in games as well as to improve visibility if used in virtual landscapes [30] or city environments. The user can benefit from extreme perspectives [11] and large FOV angles by having a reasonable survey of the scene [10] as well as a better size and depth perception [29]. Our method exploits the technique of dynamic environment mapping (in terms of cube map texturing) in combination with the programmable GPU. The separation of projection calculation and cube map texture creation enables a broad range of optimization techniques.

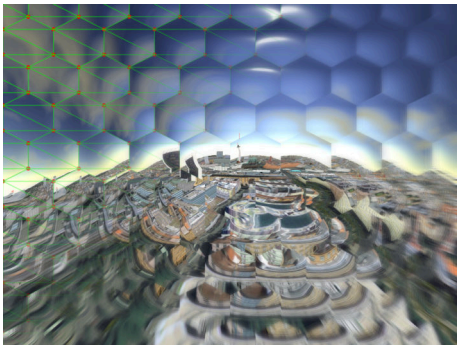

Fig. 1. Interactive visualization rendered using the concept of projection tiles.

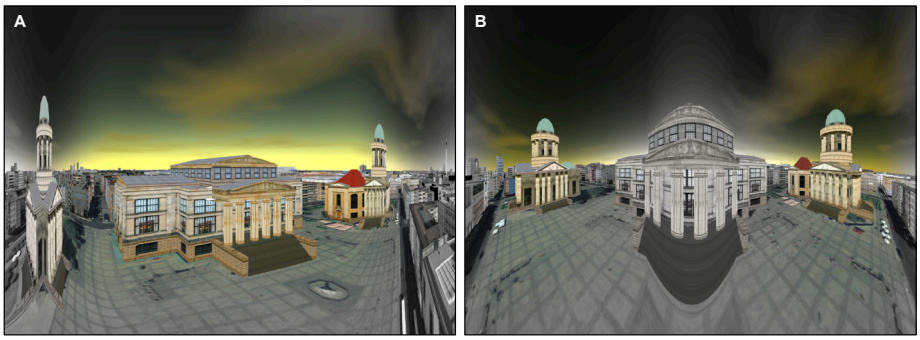

Fig. 2. Examples for combining planar and non-planar projections within a single rendering using projection tile screens. Sub-figure A is composed of a planar projection in the center and cylindrical projections left and right. Sub-figure B shows the same scene with two planar projections for each cathedral. The saturation fall-off is controlled by the respective tile features.

Existing image-based approaches for non-planar projections suffer mainly from the lack of interactive capabilities when used with complex geometric scenes such as virtual 3D city models or for large view ports. This can be explained by the trade-off between the generality of the proposed frameworks and their efficient reproduction. Furthermore, the parameterizations are complex and cannot be controlled by the user intuitively [7].

Our main contribution consists of a simple parameterizable approach to combine planar as well as non-planar projections seamlessly via so-called projection tiles. It unifies rendering techniques for creating non-planar projections, 2D screen-aligned lens effects, and 2D image-based effects. Therefore, we introduce an efficient image-based rendering concept that can be applied in a single rendering pass. Our rendering technique fully exploits current programmable consumer graphics hardware. The presented concept is easy to implement into existing rendering real-time frameworks and can be combined with other techniques that modify the image synthesis.

This work is structured in the following way: Section 2 discusses related and previous work. Chapter 3 describes the basic concept of our approach whilst Section 4 introduces a novel generalization schema for non-planar projections. Section 5 explains implementation details. Section 6 presents results and applications as well as discusses the performance and limitations of our rendering technique. Section 7 draws some conclusions and shows ideas for future work .

2 Related Work

This section gives an overview of research in the fields of SCOP projections and distortions. There is a vast amount of literature covering foundations and applications of non-planar as well as non-linear projections. In [7] a sophisticated overview is presented. To achieve distortions or special projections of the 3D scene, the pinhole camera is extended in several ways. In [1] a procedure is proposed that is based on the computation of new absolute coordinates to be transformed through an adaptive projection matrix. A

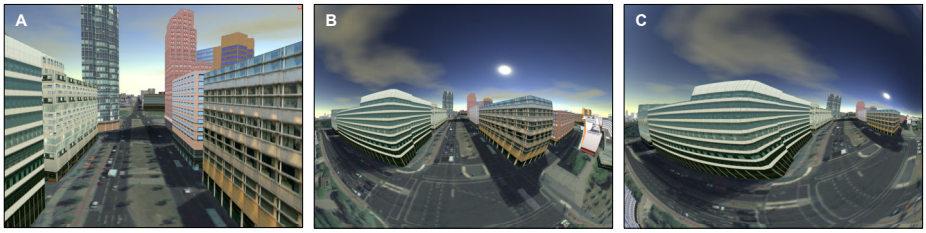

Fig. 3. Comparison between a classical perspective projection with a FOV of 45° (A) and a spherical projection with a FOV of 260° (B). Sub-figure (C) shows the same projection with an off-axis vector $O = (0.8, 0, 0)$.

flexible adaptive projection framework is described by Brosz et.al. [7] that enables the modeling of linear, non-linear, and hand-tailored artistic projections. It uses ray-casting and scan line rendering algorithm where polygonal coordinates are changed by a vertex shader. The generality of the framework makes efficient projection difficulty, especially for large scale scenes.

Distortions as sub-category of geometric registration or image warping are discussed in [16] and [12]. A warping function is applied to each pixel to determine its new color value. An image stitching approach for panorama image generation can be found in [34]. In [36] a method is demonstrated to generate environment maps from fisheye photographs. Besides the issues of nonlinear perspective deformation described in [40, 17, 41, 5, 33], we also find lens taxonomies [26, 22]. These approaches use a regular mesh textured with a 2D texture that contains the rendered scene or an image. The displacement of the mesh vertices together with the texture mapping process generates the particular distortion effect. These approaches are limited regarding the FOV which can be achieved. Carpendale researched the usage of image deformation in the context of information visualization [8]. The application of fisheye views in information visualization is discussed in [30]. Applications for view distortions in ray-tracing software are described in [15, 9].

3 Real-time Non-Planar Projections

Before focusing on the concept of projection tiles, we describe an approach that is capable of generating multiple non-planar projections in real-time (Figure 3). It is inspired by the idea described in [37]. This CPU-based technique renders six views with 90 degree FOV in each direction. Afterwards, a mapping table is used to transform these pixels into a single view according to fisheye and panorama projections. This concept can be transferred to a fully GPU accelerated implementation by using cube map textures [27] and fragment shader functionality [28]. Our approach consists of the following three components:

1. **Dynamic Environment Map**: Dynamic environment mapping [19] enables the image-based representation of the complete virtual environment for the surrounding seen from the users position. Figure 4 shows different types of environment maps. We select a cube map texture representation because it is distortion free, possesses

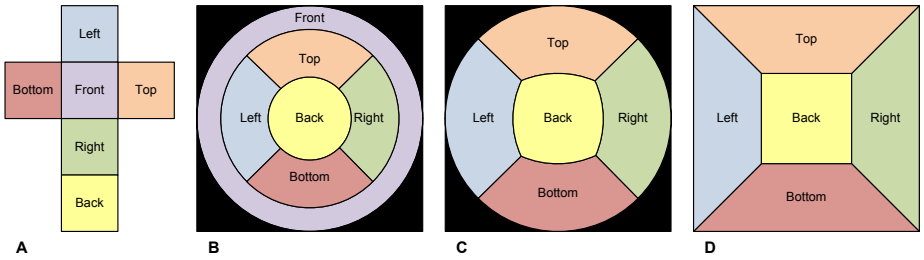

Fig. 4. Comparison between different types of environment maps. A: Cube map texture that consists of six equal-sized 2D textures. B: A Sphere map that utilizes only 78% of the available texture space and contains distorted areas (front). C: Back-paraboloid environment map. D: Pyramidal environment map as a seldom used type that fully utilizes the available 3D texture space.

an optimal texture utilization, and is a fully hardware accelerated feature [27]. A cube map texture can be constructed by using single or multi-pass rendering (see Section 5.1).

2. **Projection Canvas**: The *projection canvas* (PC) is a parametrized geometry that represents the planar area on which a particular projection is rendered on. For example, this can be screen-aligned quad that covers the whole view port or a quad which is placed in the 3D scene. Exemplary parametrization of the projection canvas are displayed in Figure 5.B and C. During rendering, graphics hardware interpolates the given parameters per fragment.
3. **Shader Functionality**: Most of the modern polygonal graphics hardware is capable of executing shader programs [20]. Our concept relies on fragment shader functionality to encapsulate the projection math by implementing a so-called *projection function* (see Section 3.1) that is evaluated for each point on the projection canvas. A projection function is a mapping of a 2D point on the parametrized PC to a 3D cube map sampling vector. During rendering, the hardware rasterizer interpolates the parameter of the PC which are then evaluated by the shader program that represents a specific projection function (Section 5.2).

This image-based concept enables the rendering of multiple non-planar projections from a single cube map texture in real-time. This can be done without any assumptions on the character of the scene, type of geometry, or rendering approach.

3.1 Projection Functions

To derive a non-planar projection of the environment captured in the cube map, a 3D cube map sampling vector $S = (x,y,z) \in \mathbb{D}^3$ for each fragment $F_{st} = (s,t) \in \mathbb{D}^2$ on the rasterized projection canvas is determined. Where $\mathbb{D} = [-1;1] \subset \mathbb{R}$ is a normalized coordinate space. Figure 5 shows the parametrization of the cube map (A) and the projection canvas (B and C). Formally, a projection function $\delta_{\mathcal{P}}(F_{st}) = S$ for a projection $\mathcal{P}$ can be defined as:

$$\delta_{\mathcal{P}} : \mathbb{D}^2 \longrightarrow \mathbb{D}^3 \qquad (s,t) \longmapsto (x,y,z) \tag{1}$$

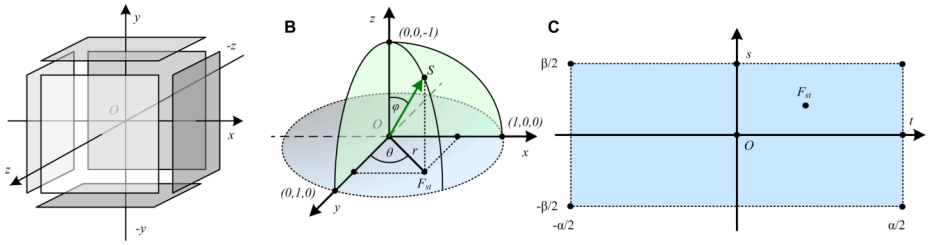

Fig. 5. Overview of the utilized coordinate systems for creating non-planar projections. A: Coordinate system and orientation of a cube map texture. B: Polar coordinates for deriving spherical projections. C: Parametrization of the projection canvas that is suitable for cylindrical projections.

Figure 6.A shows a horizontal cylindrical projection $\mathcal{C}$ that can be formulated as instance $\delta_C(F_{st}, \alpha, \beta) = S$ with an horizontal FOV of $2 \cdot \alpha$ and a vertical FOV of $2 \cdot \beta$:

$$x = \cos(s \cdot \alpha) \quad y = t \cdot \tan(\beta) \quad z = \sin(s \cdot \alpha) \tag{2}$$

Figure 5.C shows a possible parametrization of the projection canvas. Further, a spherical projection $\mathcal{S}$ with an FOV of γ can be expressed as $\delta_S(F_{st}, \gamma) = S$ with:

$$\begin{array}{ll} x = \sin(\theta) \cdot \cos(\phi) & \phi = \arctan(t, s) \\ y = \sin(\theta) \cdot \sin(\phi) & \theta = r \cdot \gamma/2 \\ z = \cos(\theta) & r = \sqrt{s^2 + t^2} \end{array} \tag{3}$$

Figure 6.B shows an resulting example of this projection while Figure 5.B displays the mapping between a point on the projection canvas (F_{st}) and cube map sampling vector S. The ideal hemisphere differs from an approximation for fisheye lenses [25]:

$$\theta = 1.411269r - 0.094389r^3 + 0.25674r^5 \tag{4}$$

3.2 Adjusting the Projection Camera Orientation

The above procedure assumes a cube-map camera orientation toward the negative z-axis. To decouple the orientation of the cube map and the projection, the sampling normal S has to be adjusted before sampling the cube map. If the cube map texture is created in the standard orientation (Figure 5.A), we would have to correct S by

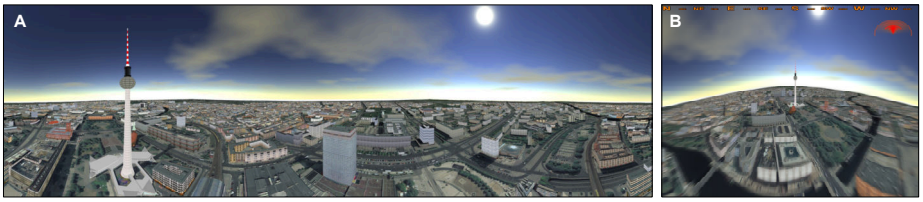

Fig. 6. Standard application examples of our rendering technique. A: Panoramic (cylindrical) projection with 360° horizontal FOV. B: Spherical projection with a FOV of 180° and applied motion blur.

transforming it with respect to the current parameters of the projection camera orientation. Let $\mathbf{C}$ be an orthonormal base constructed from the current look-to vector $L_T = (x_T, y_T, z_T) \in \mathbb{D}^3$, look-up vector $L_U = (x_U, y_U, z_U) \in \mathbb{D}^3$ and the cross product $L_C = (x_C, y_C, z_C) = L_T \times L_U$. Following this, for a projection $\mathcal{P}$, a fragment F_{st}, and a projection camera orientation $\mathbf{C}$, the final sampling vector V is calculated via:

$$V = \left(\mathbf{C} \cdot \delta_{\mathcal{P}}(F_{st} \cdot s)\right) - O \qquad \mathbf{C} = \begin{bmatrix} x_T & y_T & z_T \\ x_U & y_U & z_U \\ x_C & y_C & z_C \end{bmatrix} \Longleftrightarrow L_T \bullet L_U \neq 0 \tag{5}$$

The vector $O \in \mathbb{D}^3$ is denoted as off-axis vector [6]. Figure 3.C demonstrates an example for an off-axis projection. The scalar term $s \in \mathbb{D}$ can be interpreted as a global zooming parameter (see Section 4.1 and 6.1).

3.3 Normal-Map Optimization

If the parameters of a projection are constant, a simple optimization method can be applied. To avoid redundant shader calculations at runtime, the sampling vector S can be stored into a high-precision 32bit IEEE conformal floating-point texture map [21]. This reduces the shader execution costs for calculating a projection function to two texture look-up operations. For full screen projections, the resolution of the texture map should match the view port resolution. Figure 7 shows examples of projections (A) and their associated normal maps (B). The different value domains of normals and color are compensated for viewing. In contrast to tangent-space normal-maps, which store a normal regardless of its relative position in the tangent texture-space, we employ unit-space normal maps. This is necessary for the correct sampling of the cube map texture later on. The cube map sampling vector in unit-space N^U_{st} can be derived form a tangent-space normal N^T_{st} for each fragment F_{st} via:

$$S = \delta_{\mathcal{N}}(F_{st}) = N^U_{st} = \|N^T_{st} + (s,t,0)\| \tag{6}$$

Section 6.1 presents further application examples. This optimization technique adds an additional parametrization to our approach. Normal-maps can be stored as files using floating point images formats (http://www.openexr.org), and blending can be used to combine different normal maps to achieve mixed projections (Figure 7.B). However, this normal-map approach enables a new technique for creating and combining more complex projections using *projection tiles*.

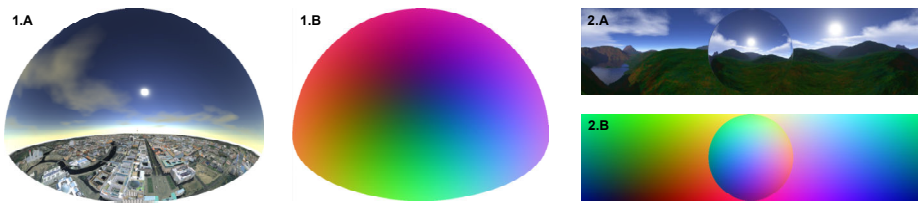

Fig. 7. Examples for normal-map optimization. Figure 1.B and 2.B show the normal map for the respective projection 2.A and 2.B. Sub-figure 2 demonstrates the result of blending two normal maps (cylindrical and spherical) to obtain a mixed projection.

4 Concept of Projections Tiles

Among others, this method enables the combination of planar projections and different non-planar derivatives as well as it facilitates the creation of custom projections which are hard to describe analytically. In this context, projection tiles are a generalization of the concept described in Section 3. Since non-planar projections can be represented as normal maps, the question arises how these textures can be created and manipulated in an efficient way. For this purpose, projection tiles provide an additional parametrization of the projection canvas. The generalization is implemented by assigning a specific view direction to each fragment of the projection canvas. Our concept is based on three components which match the programming model of current polygonal hardware rasterizers:

- **Tile Feature**: Basically, a *tile feature* (TF) describes a viewing direction for a specific point on the projection canvas. Additionally, a tile feature can be attributed with custom, user-defined attributes such as scaling and blending factors.
- **Projection Tile**: A *projection tile* (PT) is a container that consists of a finite number of tile features. A projection tile controls the interpolation of the viewing directions and the custom attributes of a tile feature.
- **Projection Tile Screen**: A *projection tile screen* (PTS) is a container for projection tiles. Together with projection tiles and tile features, it represents a parametrization of the projection canvas. At runtime, PTSs can be changed and updated. During rending, tile screens will be transformed into normal-maps in order to integrate in our framework.

One can think of this concept as an analogy to polygonal data. Here, a tile feature can be interpreted as an attributed vertex, a projection tile as a primitive, and a projection tile screen as a polygonal mesh. This design closely matches polygonal rendering hardware and thus, is easy to implement. It further facilitates the integration into existing digital content creation (DCC) pipelines by reading and writing 3D meshes. Thereby, custom feature attributes can be encoded as texture coordinate values.

4.1 Tile Features

To enable an intuitive way to describe a TF, we have chosen spherical polar coordinates ϕ and θ to express the view direction instead of using a normalized direction vector. We define a tile feature E as a 6-tupel:

$$E = (x, y, \phi, \theta, s, f) \qquad x, y, s, f \in [0;1] \subset \mathbb{R} \qquad \phi, \theta \in [-360;360] \subset \mathbb{R} \qquad (7)$$

that contains mainly the 2D feature position (x, y) and the particular horizontal and vertical view angles (ϕ, θ). The parameter s is a scaling factor while the variable f can be used to bind custom parameters to each feature. For example, Figure 2 demonstrates this by adjusting the image saturation according to the value of f. The universe of all tile features is denoted as $\mathcal{E}$.

4.2 Projection Tiles & Projection Tile Screens

A projection tile defines the area of a specific projection in relation to the projection canvas and consists of a number of tile features $E_i \in \mathcal{E}$. Projection tiles are organized in a *projection tile screen* (PTS) (Figure 8.A) which is represented by a specific tile set $\mathcal{T}$. We have experimented with two different types of grouping tile features: rectangular and triangular shaped projection tiles. We present both types, since both have different properties and features which have proven themselves to be beneficial:

Rectangular Projection Tiles (RPT): A *rectangular projection tile* consists of four tile features that are organized in a regularly structured rectangular projection tile screen (RPTS) $\mathcal{T}_{mn}^{RPT}$ with:

$$RPT_{kl} = (E_{(k,l)}, E_{(k+1,l)}, E_{(k+1,l+1)}, E_{(k,l+1)}) \quad E_{ij} \in \mathcal{E} \quad \mathcal{T}_{mn}^{RPT} = \begin{bmatrix} E_{0n} & \cdots & E_{mn} \\ \vdots & \ddots & \vdots \\ E_{00} & \cdots & E_{m0} \end{bmatrix} \tag{8}$$

Using rectangular tiles introduces a number of disadvantages. First of all, not every tile shape can be represented with four tile features. The structure of the rectangular projection tile screen $\mathcal{T}_{mn}^{RPT}$ allows no non-continuous transitions between projection tiles. The regular structure of RPTS and its associated RPT enable an easy direct manipulation.

Triangular Projection Tiles (TPT): To overcome the drawbacks of the RPTs, we used *triangular projection tiles* TPT:

$$TPT_i = (E_0, E_1, E_2), \qquad E_i \in \mathcal{E}, \qquad \mathcal{T}_n^{TPT} = TPT_0, \ldots, TPT_n \tag{9}$$

With TPTs, the user is able to control the triangulation of the projection canvas directly. This degree of freedom enables the usage of 2D lenses as shown on Figure 11.B. TPTs also introduce non-continuous transitions between the tiles but are difficult to manipulate directly by the user.

4.3 Mapping Projection-Tile-Screens to Normal-Maps

The concept of projection tiles can be integrated in our existing framework by using the normal-map optimization method described in Section 3.3. Therefore, we have to define a mapping between a PTS and a normal map. Figure 8 gives an overview of the complete process. A PTS (RPTS or TPTS) is transformed into a normal map (Figure 8.B) by rendering all of its respective projection tiles (RPT or TPT) into an off-screen 2D texture-target using *render-to-texture* [39]. Thereby, the tile feature components x and y are interpreted as the 2D vertices of a quad or triangle in standard orthographic parallel projection [38].

The angles and user defined attributes are encoded into per-vertex texture coordinates. This avoids any domain-specific encoding of these values. The hyperbolic interpolation [3] between these values is then performed by graphics hardware. A fragment

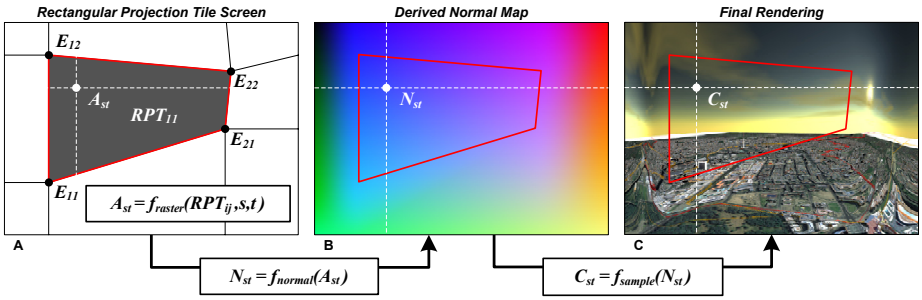

Fig. 8. Elaborated conceptual pipeline to render combinations of planar and non-planar projections defined by rectangular projection tiles. A projection tile screen (A) is transformed into a normal map (B) that contains the cube map sampling vectors to create the final rendering (C).

shader converts the horizontal and vertical angles $A_{st} = (\phi_{st}, \theta_{st})$, obtained for each fragment F_{st}, into a respective normal N_{st} and outputs it into the texture target. The normal vector can be calculated by:

$$N_{st} = \delta_{\mathcal{PTS}}(F_{st}) = \left\| \mathbf{R}_x(\theta_{st}) \cdot \left(\mathbf{R}_y(\phi_{st}) \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) + \begin{bmatrix} s \\ t \\ 0 \end{bmatrix} \right\| \tag{10}$$

Where $\mathbf{R}_x$ and $\mathbf{R}_y$ denote the 3D rotation matrices which transform the base normal $(0,0,1)$ around the respective x and y axis. For each N_{st} in the resulting normal map a sampling vector S can be calculated by setting $\delta_{\mathcal{PTS}}(F_{st} \cdot s_{st}) = N_{st}$. The result is shown in Figure 8.C.

5 Implementation Details

The implementation is designed to meet the requirements of real-time visualization of large, geometrical complex scenes. An exemplary implementation was done by using OpenGL [31] in combination with the OpenGL shading language (GLSL) [20]. It uses framebuffer objects, floating point textures, and mip-mapped cube maps for dynamic texturing [18]. We suggest a two-phase rendering process: First, create or update the dynamic cube map and Second: render the projection canvas while applying the respective projection. This section briefly describes both phases.

5.1 Cube Map Creation

The creation and update of dynamic cube map textures [14] is an essential part of our concept. This creation process can be implemented using single-pass or multi-pass rendering approaches:

– **Single-Pass Approach:** On current hardware, it is possible to create cube map textures within a single rendering pass by utilizing *geometry shaders* and *layered-rendering*[24]. This so called *render-to-cube-map* technique duplicates each input

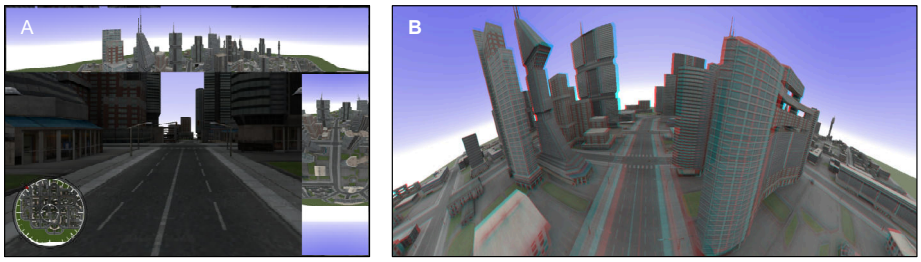

Fig. 9. Multiple projections rendered with our framework. A: Two full 360° horizontal and vertical cylindrical projections embedded as overlay onto a standard perspective projection. B: An interactive, directional, stereoscopic image of a cylindrical projection with a horizontal FOV of 260°.

triangle six times and applies a separate model-view transformation for each face of the cube map texture. Each of the duplicated triangle is directed to the respective layer of a layered render target [4].

– **Multi-Pass Approach:** One can create a cube map texture using multi-pass rendering in combination with *render-to-texture* [13]. Given a reference camera position, we can construct six local virtual cameras with a FOV of 90 degrees, an aspect ratio of 1, and render the scene into the respective cube map texture targets. There are two alternatives to construct these virtual local cameras: 1) by rotating the reference camera or 2) by using fixed cube map orientation (Figure 5.A). The latter demands for a look-to correction as described in Section 3.2 and is necessary for a simple implementation of projection tiles.

Both approaches differs with respect to runtime performance and integration costs. Figure 10 shows a runtime comparison of both approaches for two models of different polygon count (Model 1: 41032, Model 2: 46060 vertices). The test application does not utilize the second CPU core. The comparison shows the advantage of the single-pass cube map creation compared to the multi-pass approach. Rendering within a single pass has a main advantage: The scene has be traversed only once. This results in fewer state changes per frame. So, fast cube map creation enables the usage of more than one dynamically created cube map. For an exemplary application, Figure 9.B shows a stereoscopic rendering of a 260° cylindrical projection.

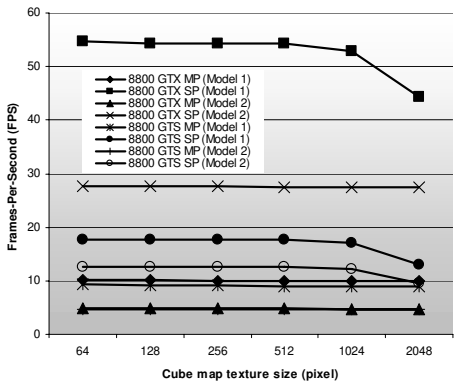

Fig. 10. Run-time performance for multi-pass (MP) and single-pass (SP) cube map creation techniques measured with our rendering framework.

Using the single-pass creation methods introduces also problems and drawbacks. In contrast to the multi-pass approach, occlusion culling algorithms cannot be integrated straight forward. Further, a successful application to out-of-core rendering concepts is

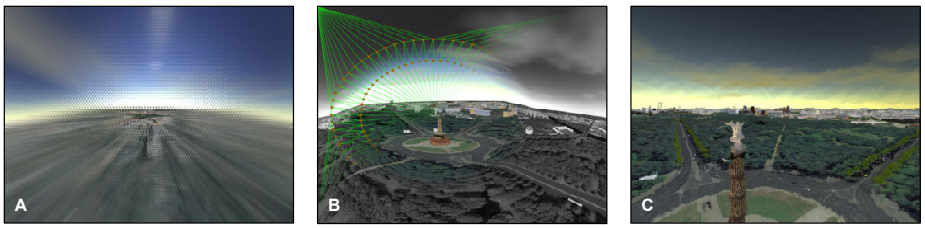

Fig. 11. Application examples created with our rendering technique. A: Shows a compound-eye projection tiles screen with approximately 30,000 triangular projection tiles. B: A screen-aligned 2D magnification lens which is embedded within a fisheye projection with 180° FOV. C: Image-distortion based on a normal map.

questionable. Furthermore, using geometry shader introduce also a conceptual problem. Current shading languages [20] are not designed to combine different functional decomposed shader into a single one. A programmer of a complex visualization framework has to choose between multiple shader variations or a system that supports the combination in an automated manner [35].

5.2 Applying Projections

Given a generated cube map, a particular projection is applied in a post-processing pass subsequent to the cube map creation pass(es). For rendering a full-screen non-planar projection the following three steps are performed:

1. Setup a standard 2D orthographic projection. The camera is set to the standard orientation with a look-to vector $L_T = (0,0,-1)$.
2. Activate a specific fragment program that implements the mathematical concepts as described in Sections 3.1 and 4. The shader program performs cube map texture lookups or outputs the calculated normal vectors for later re-use.
3. Render a screen-aligned quad (projection canvas) with standard texture coordinates that covers the entire view port.

6 Experimental Results

6.1 Application Examples

Projection tiles enable a broad range of applications for the interactive visualization of large scale datasets, such as 3D virtual city and landscape models. They can be used to create standard non-planar projections (Figure 6) and can easily control its variations. This is the standard use-case for our rendering framework. These projections can be applied to non-planar projection surfaces. Figure 12 shows such application to a projector systems. Here, the pincushion distortion is compensated by the projectors.

Further, projection tiles facilitate different combinations of planar and non-planar projections (Figure 2) and thus, enable distortion-based visualization techniques for virtual environments. Triangulated projection tiles support the embedding of 2D screen aligned lenses into non-planar projections. Figure 11.B shows an example of a zooming lens. The lens position can be changed at runtime while the PTS is adapted per frame. To avoid possible under-sampling artifacts, a sufficient resolution of the cube map must be available. Figure 1 shows an application that utilize projection tiles to create a complex visualization which approximates a compound-eye like view on a virtual environment. Our framework also supports image-warping for planar and non-planar projections based on user defined normal maps. This functionality is based on the normal map optimization described in Section 3.3. It enables post processing effects water on lenses or the simulation of shower door effects (Figure 11.C).

Fig. 12. Cylindrical projector system (A) used with our rendering technique (B).

6.2 Problems & Limitations

The main drawback of our concept is a possible lack of image quality compared to other approaches. This can have numerous reasons but is mainly caused by texture under- and oversampling artifacts that can results in blurred output images. To avoid these artifacts, the optimal resolution of the cube map texture mus be adapted to the resolution of the view port and the used projection or projection tile screen. This is a problem for current hardware generations since rendering to texture target perform sub-optimal for resolutions which are not power-of-two. We observed that for high horizontal and vertical FOV, a cube map resolution of 1024^2 pixels is sufficient within reasonable view port size (1600x1200). Further, the quality of the output images for projection tiles screen depend on the resolution of the PTS. If the resolution is too low, the feature interpolation can cause artifacts for tiles with acute angles.

7 Conclusions & Outlook

This work presented the concept of projection tile screens, a generalization of single-center projections and image distortions that is applicable in real-time especially for large scenes. It enables the efficient creation and combination of planar as well as non-planar single center projections, 2D screen-aligned lenses with arbitrary shapes [32, 8, 2], image warping and image distortions. The concept is based on dynamic cube maps and programmable hardware features. We further present a comparative performance evaluation for creating dynamic cube maps using single-pass and multi-pass approaches. Our future work focuses on possible applications of projection tiles screens, especially the interactive modification of projection tile screens using in-space authoring tools. Further, we heading to improve rendering quality. We are particularly interested in reproducing other SCOP projections as described by [23].

Acknowledgments

I'd like to thank Haik Lorenz for fruitful discussions on this topic. This work has been funded by the German Federal Ministry of Education and Research (BMBF) as part of the InnoProfile research group '3D Geoinformation' (www.3dgi.de).

References

1. Salvador Bayarri. Computing Non-Planar Perspectives in Real Time. *Computers & Graphics*, 19(3):431–440, 1995.
2. Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and Magic Lenses: The See-Through Interface. In *SIGGRAPH*, pages 73–80. ACM Press, 1993.
3. Jim Blinn. Hyperbolic Interpolation. *IEEE Computer Graphics and Applications Staff*, 12(4):89–94, 1992.
4. David Blythe. The Direct3D 10 System. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 724–734, New York, NY, USA, 2006. ACM Press.
5. Paul Bourke. Nonlinear Lens Distortion, August 2000.
6. Paul Bourke. Offaxis Fisheye Projection, October 2004.
7. John Brosz, Faramarz F. Samavati, M. Sheelagh, T. Carpendale, and Mario Costa Sousa. Single Camera Flexible Projection. In *NPAR '07: Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, pages 33–42, New York, NY, USA, 2007. ACM Press.
8. M. S. T. Carpendale and Catherine Montagnese. A Framework for Unifying Presentation Space. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 61–70, New York, NY, USA, 2001. ACM Press.
9. Patrick Coleman and Karan Singh. RYAN: Rendering Your Animation Nonlinearly projected. In *NPAR*, 2004.
10. Georg Glaeser and Eduard Gröller. Fast Generation of Curved Perspectives for Ultra-Wide-Angle Lenses in VR Applications. *The Visual Computer*, 15(7/8):365–376, 1999.
11. Georg Glaeserm. Extreme and Subjective Perspectives. In *Topics in Algebra, Analysis and Geometry*, pages 39–51, BPR Mdiatancsad BT/Budapest, 1999.
12. C.A. Glasbey and K.V. Mardia. A Review of Image Warping Methods. *Journal of Applied Statistics*, 25:155–171, 1989.
13. Dominik Göddeke. Playing Ping Pong with Render-To-Texture. Technical report, University of Dortmund, Germany, 2005.
14. Ned Greene. Environment Mapping and other Applications of World Pojections. *IEEE Comput. Graph. Appl.*, 6(11):21–29, 1986.
15. Meister Eduard Gröller and Pietro Acquisto. A Distortion Camera for Ray Tracing. In Conner, Hernandez, Murthy, and Power, editors, *Visualization and Intelligent Design in Engineering and Architecture*. Elsevier Science Publishers, April 1993.
16. Andreas Gustafsson. Interactive Image Warping. Master's thesis, Faculty of Information Technology, 1993.
17. Bao H., Ying J., and Peng Q. Non-Linear View Interpolation. In *The Journal of Visualization and Computer Animation*, volume 10, pages 233–241(9). John Wiley & Sons, Ltd., October/December 1999.
18. Mark Harris. Dynamic Texturing. NVIDIA Corporation, May 2004.
19. Wolfgang Heidrich and Hans-Peter Seidel. View-independent Environment Maps. In *HWWS '98: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 39–ff., New York, NY, USA, 1998. ACM Press.

20. John Kessenich. *The OpenGL Shading Language Version 1.20*, 59 edition, April 2004.
21. Mark J. Kilgard. NVIDIA OpenGL Extension Specifications. Technical report, NVIDIA Corporation, May 19, 2004.
22. Y.K. Leung and M.D. Apperley. A Review and Taxonomy of Distortion-Oriented Presentation Techniques. *ACM Transactions on Computer-Human Interaction*, 1:126–160, 1994.
23. F. Margaret. Perspective Projection: the Wrong Imaging Model, 1995.
24. Microsoft. Direct3D 10 Programming Guide Excerpts. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, pages 369–446, New York, NY, USA, 2007. ACM Press.
25. Max L. Nelson. Computer Graphics Distortion for IMAX and OMNIMAX Projection. In *Nicograph 83MaxNicograph1983*, pages 137–159, December Nicograph 1983.
26. Petra Neumann and Sheelagh Carpendale. Taxonomy for Discrete Lenses. Technical Report 2003-734-37, Department of Computer Science, University of Calgary, December 2003.
27. NVIDIA. OpenGL Cube Map Texturing, May 2004.
28. NVIDIA. *NVIDIA GPU Programming Guide*. NVIDIA Corporation, 2.4.0 edition, August 2005.
29. Jennifer A. Polack-Wahl, Les A. Piegl, and Marc L. Carter. Perception of Images Using Cylindrical Mapping. *The Visual Computer*, 13(4):155–167, 1997.
30. Wolf-Dieter Rase. Fischauge-Projektionen als kartographische Lupen. In F. Dollinger and J.Strobl, editors, *Angewandte Geographische Informationsverarbeitung*, volume IX of *Salzburger Geographische Materialien*. Selbstverlag des Instituts für Geographie der Universität Salzburg, 1997.
31. Mark Segal and Kurt Akeley. *The OpenGL Graphics System: A Specification, Version 2.0*, October 2004.
32. Martin Spindler, Marco Bubke, Tobias Germer, and Thomas Strothotte. Camera Textures. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 295–302, New York, NY, USA, 2006. ACM Press.
33. R. Swaminathan, M.D. Grossberg, and S.K. Nayar. A Perspective on Distortions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume II, pages 594–601, Jun 2003.
34. Richard Szeliski and Heung-Yeung Shum. Creating Full View Panoramic Image Mosaics and Environment Maps. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 251–258, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.
35. Matthias Trapp and Jürgen Döllner. A generalization approach for 3d viewing deformations of single-center projections. In Nuno Jardim Nunes José Braz and Joao Madeiras Pereira, editors, *GRAPP 2008 - International Conference on Computer Graphics Theory and Applications*, number 3, pages 162–170. INSTICC Press, January 2008.
36. Ken Turkowski. Making Environment Maps from Fisheye Photographs, 1999.
37. Wouter van Oortmerssen. FisheyeQuake/PanQuake, January 2002.
38. Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
39. Chris Wynn. OpenGL Render-to-Texture. In *GDC*. NVIDIA Corporation, October 2002.
40. Yonggao Yang, Jim X. Chen, and Mohsen Beheshti. Nonlinear Perspective Projections and Magic Lenses: 3D View Deformation. *IEEE Computer Graphics and Applications*, pages 76–84, 2005.
41. Yonggao Yang, Jim X. Chen, Woosung Kim, , and Changjin Kee. Nonlinear Pojection: Using Deformations in 3D Viewing. In Jim X. Chen, editor, *Visualization Corner*, pages 54–59. IEEE, March/April 2003.